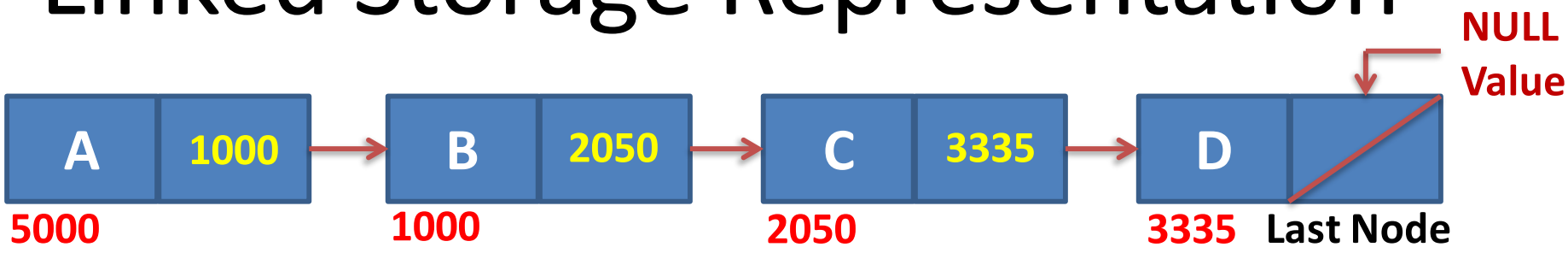
Linked List

Linked Storage Representation

- There are many applications where **sequential allocation** method is **unacceptable** because of following characteristics
  - **Unpredictable storage** requirement
  - **Extensive manipulation** of stored data
- One method of obtaining the address of node is to store address in computer's main memory, we refer this addressing mode as **pointer of link addressing**.
- A simple way to represent a linear list is to expand each node to contain a link or pointer to the next node. This representation is called one-way chain or Singly Linked Linear List.

Linked Storage Representation

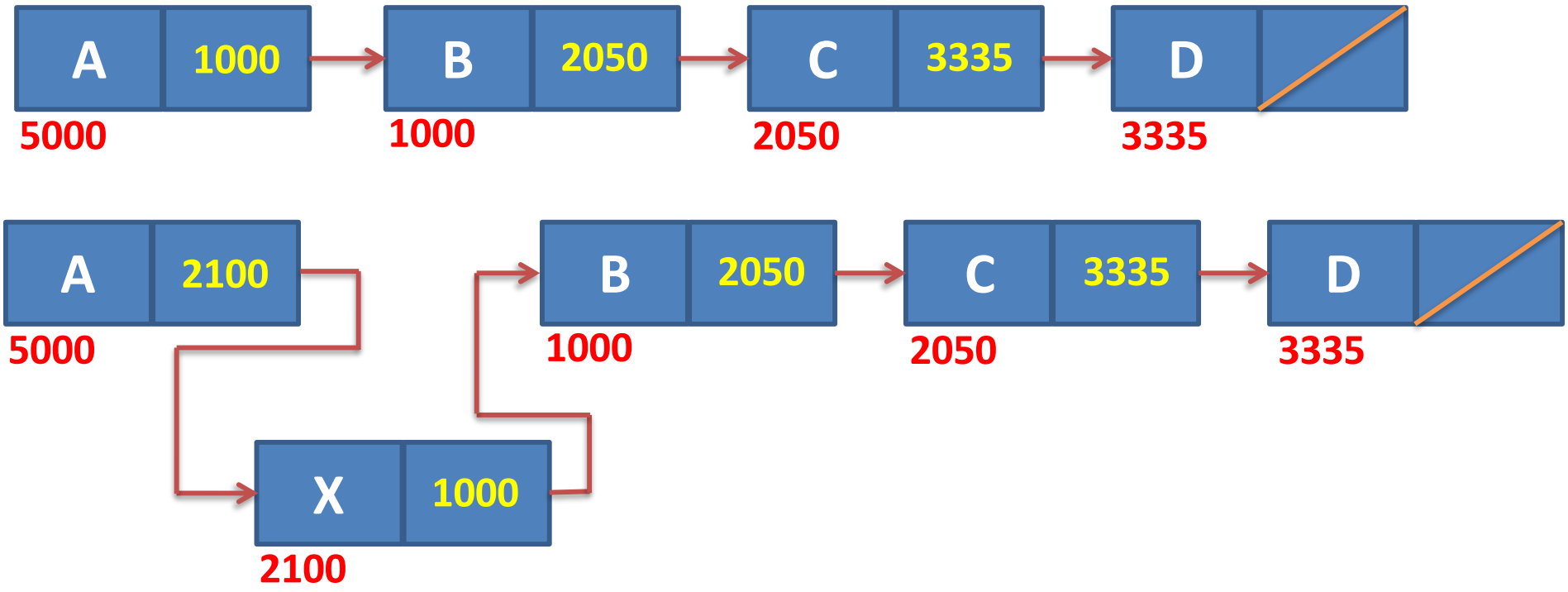


A linked List

- The linked allocation method of storage can result in both efficient use of computer storage and computer time.
  - A linked list is a **non-sequential collection** of data items.
  - Each **node** is **divided** into **two parts**, the **first part** represents the **information** of the element and the **second part** contains the **address of the next mode**.
  - The **last node** of the list does not have successor node, so **null value** is stored as the address.
  - It is possible for a list to have no nodes at all, such a list is called empty list.

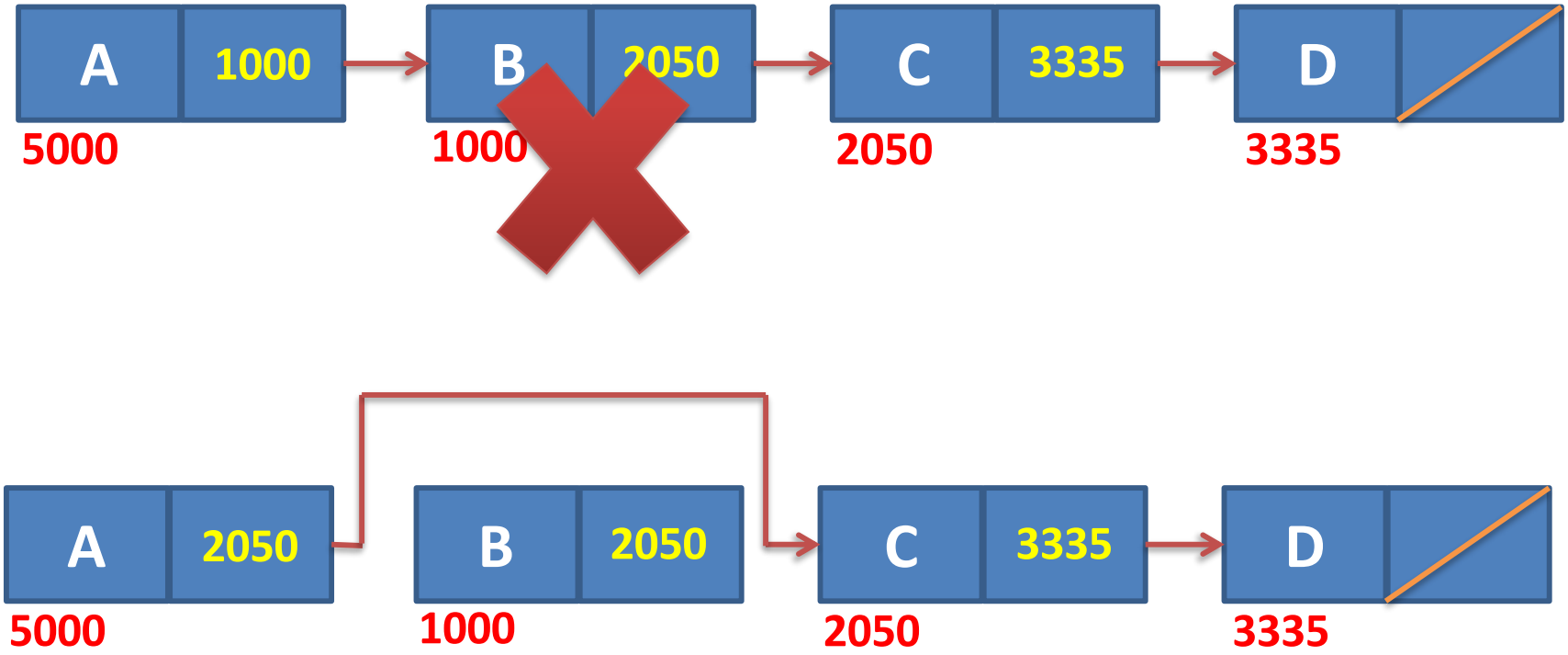
Pros & Cons of Linked Allocation

- **Insertion Operation**
  - we have an n elements in list and it is required to insert a new element between the first and second element, what to do with sequential allocation & linked allocation?
  - Insertion operation is more efficient in Linked allocation.



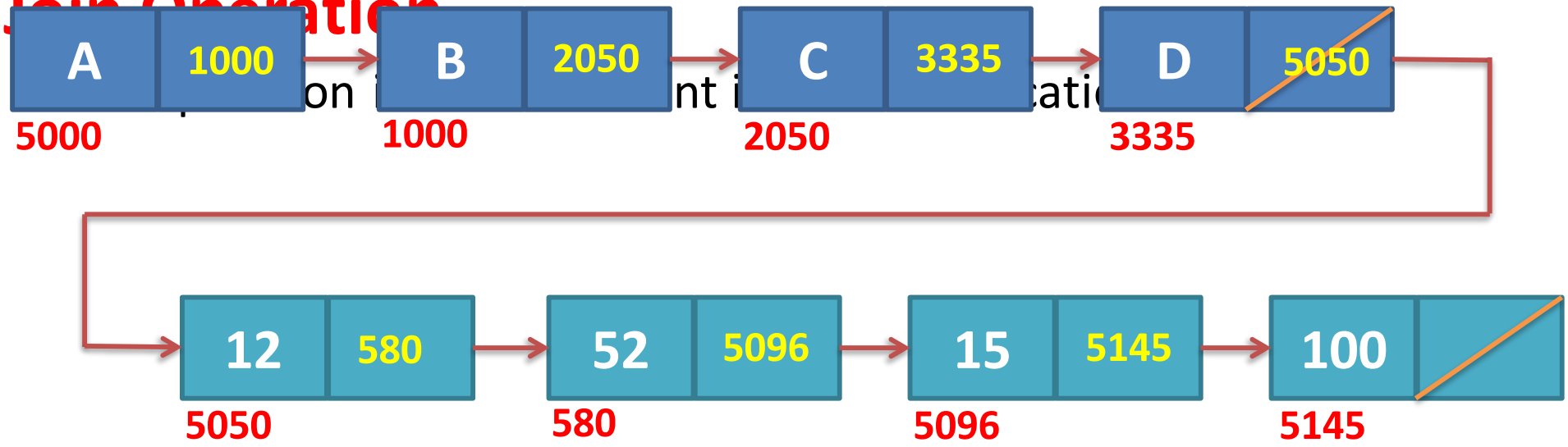
Pros & Cons of Linked Allocation

- **Deletion Operation**
  - Deletion operation is more efficient in Linked Allocation



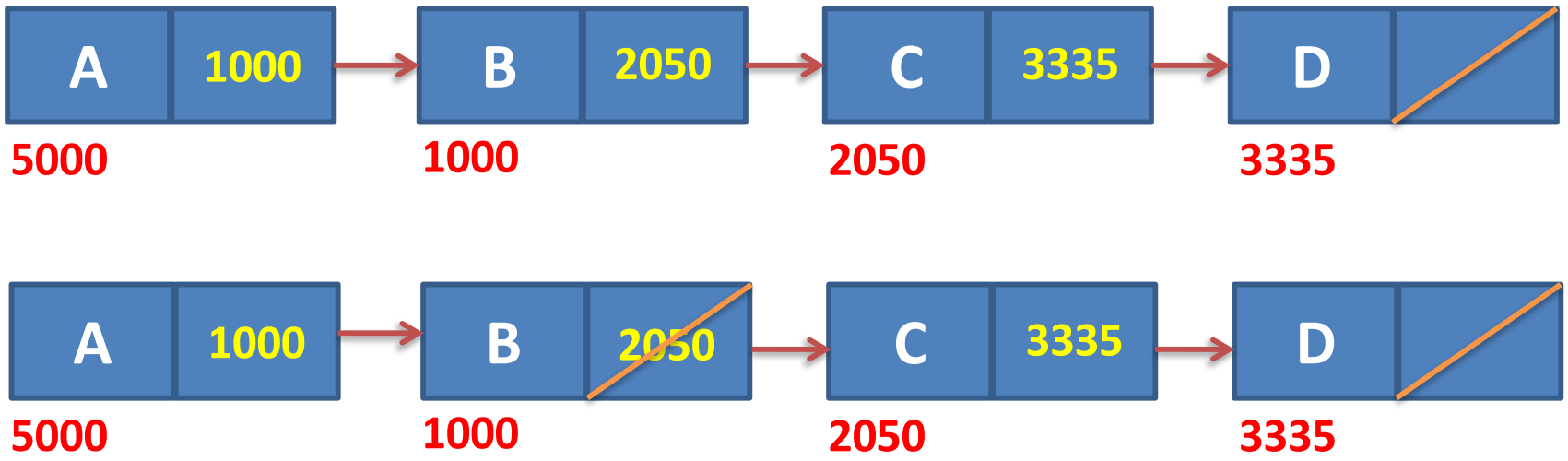
Pros & Cons of Linked Allocation

- **Search Operation**
  - **If particular node in the list is required**, it is necessary to follow links from the first node onwards until the desired node is found, in this situation **it is more time consuming** to go through linked list than a sequential list.
  - Search operation is more time consuming in Linked Allocation.
- **Join Operation**



Pros & Cons of Linked Allocation

- Split Operation
  - Split operation is more efficient in Linked Allocation



Pros & Cons of Linked Allocation

- Linked list require **more memory** compared to array because along with value it stores pointer to next node.
- Linked lists are among the simplest and most common data structures. They can be used to implement other data structures like stacks, queues, and symbolic expressions, etc…

Operations & Type of Linked List

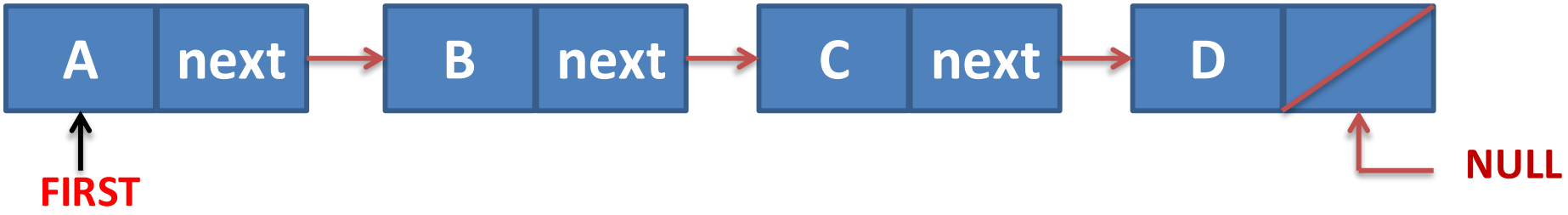
Operations on Linked List

- Insert
  - Insert at first position
  - Insert at last position
  - Insert into ordered list
- Delete
- Traverse list (Print list)
- Copy linked list

Types of Linked List

- Singly Linked List
- Circular Linked List
- Doubly Linked List

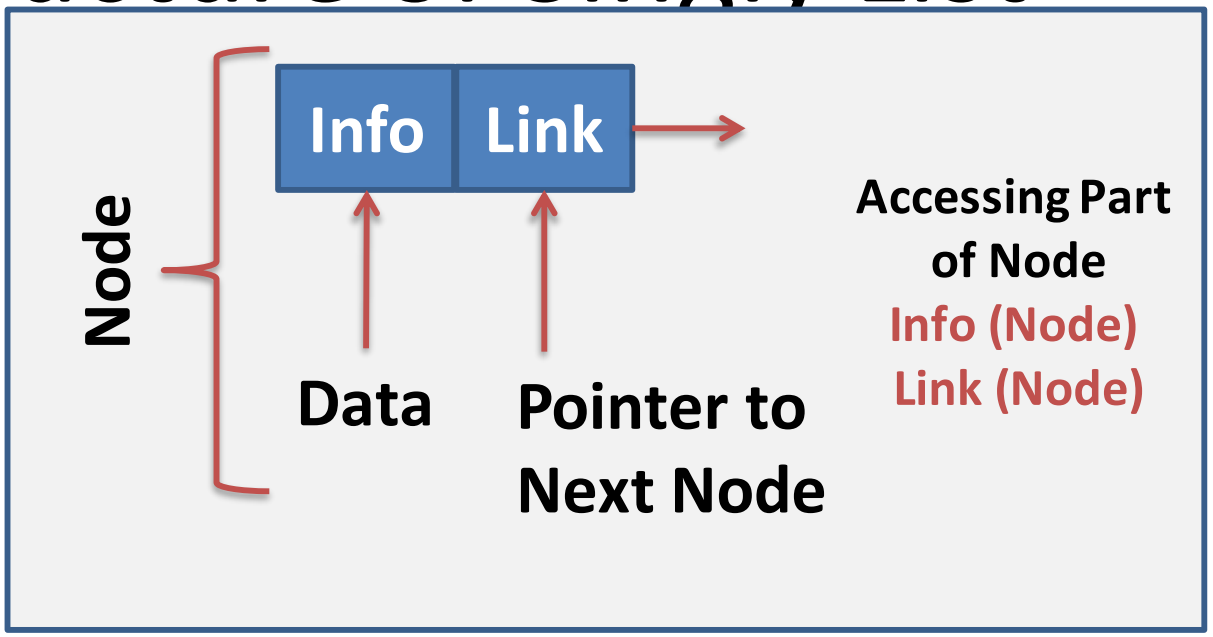
Singly Linked List



- It is basic type of linked list.
- Each node contains data and pointer to next node.
- Last node's pointer is null.
- First node address is available with pointer variable **FIRST**.
- **Limitation** of singly linked list is **we can traverse only in one direction**, forward direction.

Node Structure of Singly List

Typical Node



C Structure to represent a node

```
struct node
{
        int info;
        struct node *link;
};
```

Algorithms for singly linked list

1. Insert at first position
2. Insert at last position
3. Insert in Ordered Linked list
4. Delete Element
5. Copy Linked List

Singly Link List

Data Structure for Singly Link List:

```
struct node{
        int data;
        struct node *next;
} *start, *new1,*ptr;
```

start POINTER WILL ALWAYS POINT TO THE FIRST ELEMENT OF THE LINK LIST

start

next POINTER HAS ADDRESS OF NEXT ELEMENT

new1

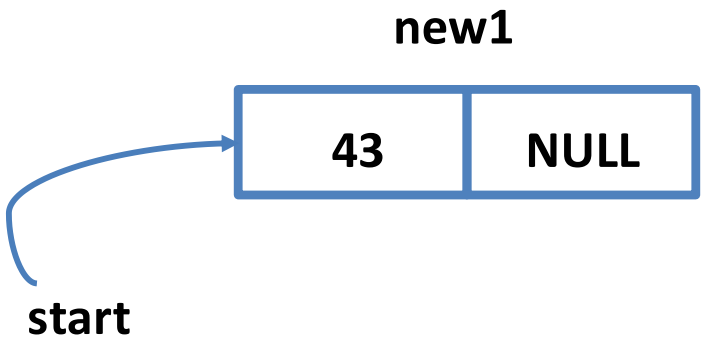
new1 and ptr are TEMPORARY POINTER AND They WILL be Used TO TAKE INPUT FROM USER, AND DEFAULT NEXT VALUE IS NULL. Like new1->next = null

Creation of first node in singly link list

```
struct node
{
        int d;
        struct node *next;
}*start=NULL, *new1;
```

- new1=(struct node *)malloc(sizeof(struct node *));
- new1->next=NULL;
- printf("\nenter element vlaue:\n");
- scanf("%d",&new1->d);

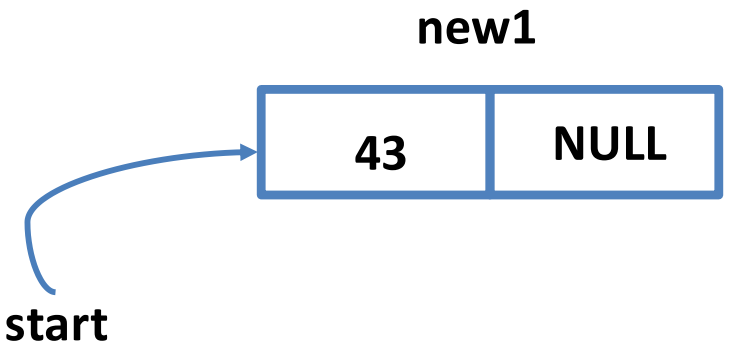
Start=new1;



Creation of singly link list

```
void create()
{
        char ch;
        do
        { new1=(struct node *)malloc(sizeof(struct node *));
          new1->next=NULL;
          printf("\nenter element vlaue:\n");
          scanf("%d",&new1->d);
          if(start==NULL)
          {
            start=new1;
          }
          else
          {
                ptr=start;
                while(ptr>next!=NULL)
                        ptr=ptr->next;

                ptr->next=new1;
          }
        printf("\nenter choice(press n for exit\n");
        ch=getche();
        }while(ch!='n');
}
```

Initially start=null, so condition is true

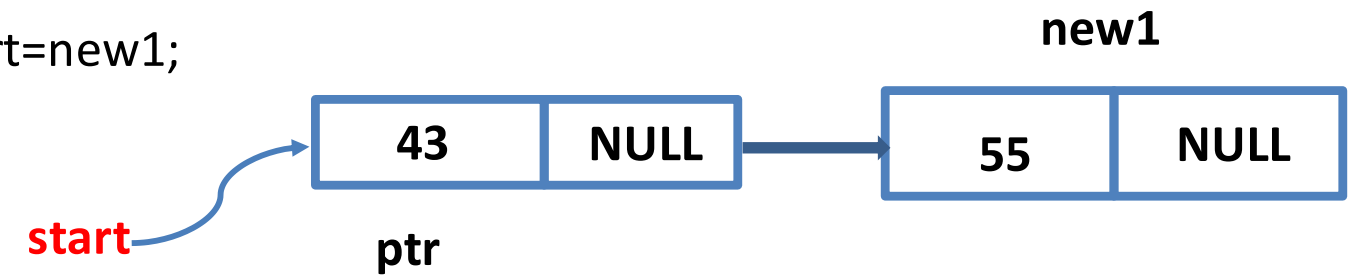


Creation of singly link list

```
void create()
{
        char ch;
        do
        { new1=(struct node *)malloc(sizeof(struct node *));
          new1->next=NULL;
          printf("\nenter element vlaue:\n");
          scanf("%d",&new1->d);
          if(start==NULL)
          {
                start=new1;
          }
          else
          {
            ptr=start;
                while(ptr->next!=NULL)
                        ptr=ptr->next;
                ptr->next=new1;
          }
        printf("\nenter choice(press n for exit\n");
        ch=getche();
        }while(ch!='n');
}
```

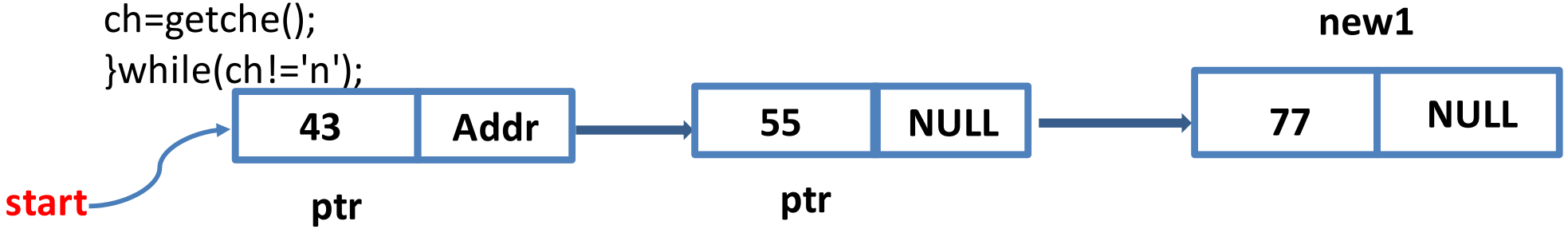
Condition is false

Condition is false



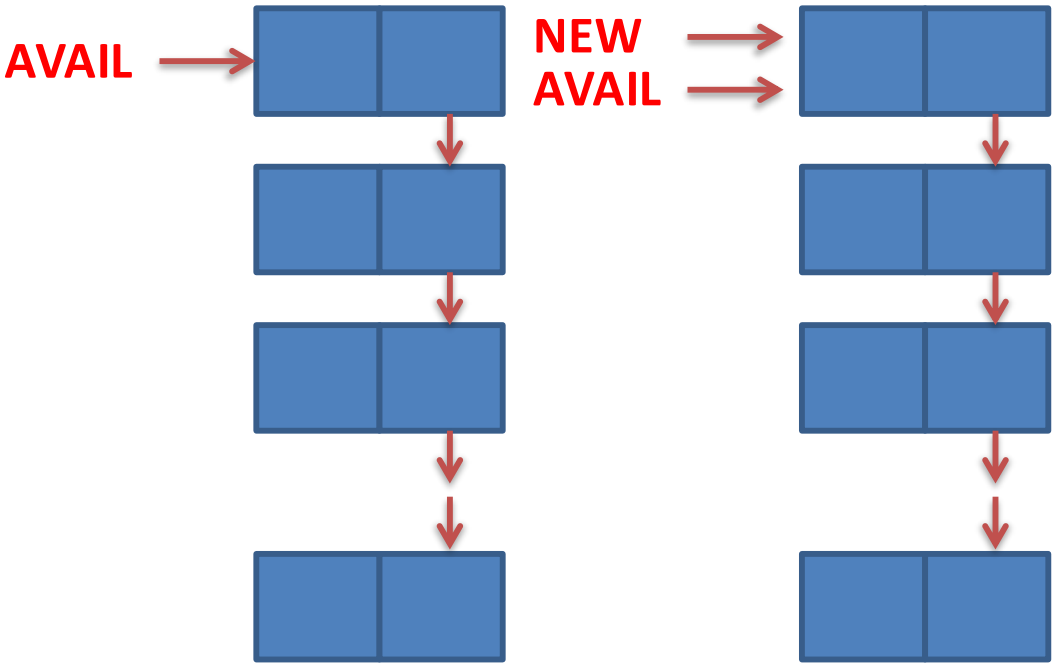
Creation of singly link list

```
void create()
{
        char ch;
        do
        { new1=(struct node *)malloc(sizeof(struct node *));
          new1->next=NULL;
          printf("\nenter element vlaue:\n");
          scanf("%d",&new1->d);
          if(start==NULL)        Condition is false
          {
                start=new1;
          }
          else
          {
                ptr=start;
                while(ptr->next!=NULL)    Condition is false
                      ptr=ptr->next;
                ptr->next=new1;
          }
        printf("\nenter choice(press n for exit\n");
        ch=getche();
        }while(ch!='n');
}
```



Availability Stack

- A **pool** or list **of free nodes**, which we refer to as the **availability stack** is maintained in conjunction with linked allocation.
- Whenever a node is to be inserted in a list, a free node is taken from the availability stack and linked to the new list.
- On other end, the deleted node from the list is added to the availability stack.



Check for free node in Availability Stack

IF AVAIL is NULL
THEN Write('Availability Stack Underflow')
Return

Obtain Address of next free node

NEW ← AVAIL

Remove free node from Availability Stack

AVAIL ← LINK(AVAIL)

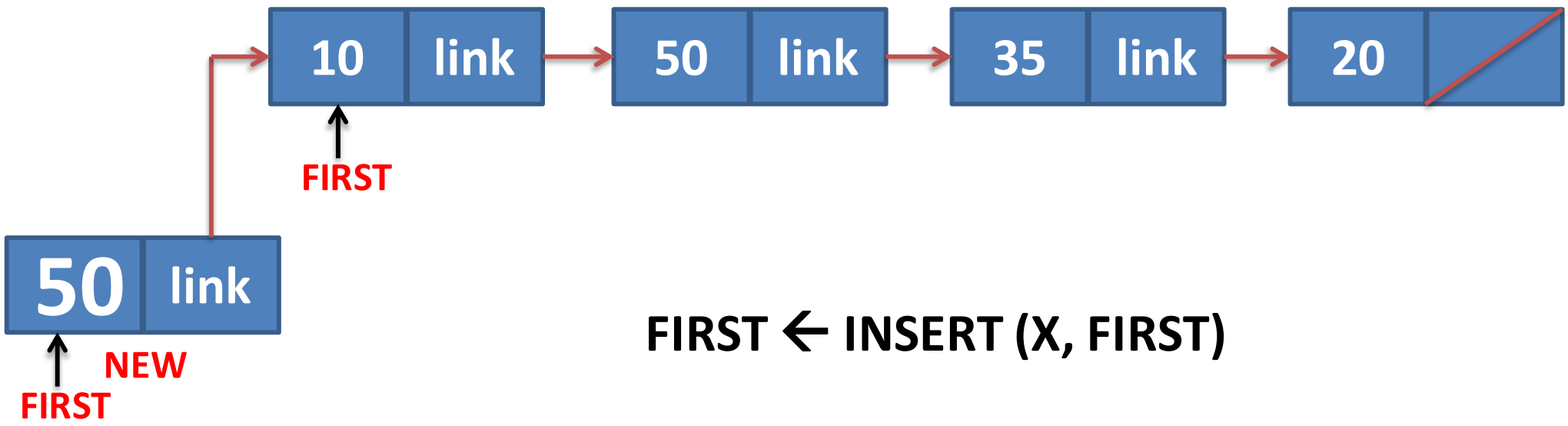
Function: INSERT(X,First)

- This function **inserts a new node at the first position** of Singly linked list.
- This function returns address of **FIRST** node.
- **X** is a new element to be inserted.
- **FIRST** is a **pointer to the first element** of a Singly linked linear list.
- Typical node contains **INFO** and **LINK** fields.
- **AVAIL** is a pointer to the top element of the availability stack.
- **NEW** is a temporary pointer variable.

Function: INSERT(X,FIRST) Cont…

```
1. [Underflow?]
   IF     AVAIL = NULL
   Then   Write ("Availability Stack Underflow")
          Return(FIRST)
2. [Obtain address of next free Node]
    NEW ← AVAIL
3. [Remove free node from availability Stack]
    AVAIL ← LINK(AVAIL)
4. [Initialize fields of new node and
   its link to the list]
    INFO(NEW) ← X
    LINK (NEW) ← FIRST
5. [Return address of new node]
    Return (NEW)
```

Example: INSERT(50,FIRST)



FIRST ← INSERT (X, FIRST)

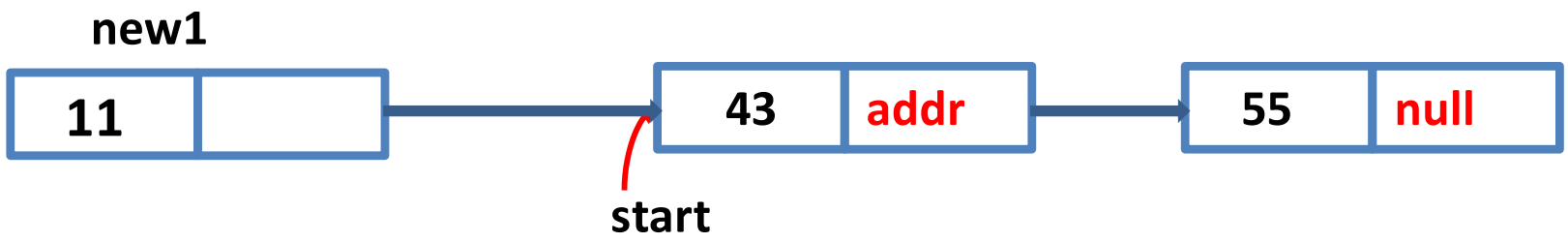
```
4. [Initialize fields of new node and
   its link to the list]
    INFO(NEW) ← X
    LINK (NEW) ← FIRST
5. [Return address of new node]
    Return (NEW)
```

Insert node at beginning of the given singly link list

```
void insertbegin()
{
    struct node *new1;
    new1=(struct node *)malloc(sizeof(struct node));
    printf(" enter element \n");
    scanf("%d",&new1->d);
    if(start==NULL)
   {
            start=new1;
            new1->next=NULL;
    }
     else
    {
            new1->next=start;
            start=new1;
     }
}
```

Condition is false

Given existing singly link list



Function: INSEND(X, FIRST)

- This function **inserts** a new node at the **last position** of linked list.
- This function returns address of **FIRST** node.
- **X** is a new element to be inserted.
- **FIRST** is a **pointer to the first element** of a Singly linked linear list.
- Typical node contains **INFO** and **LINK** fields.
- **AVAIL** is a pointer to the top element of the availability stack.
- **NEW** is a temporary pointer variable.

Function: INSEND(X, First) Cont…

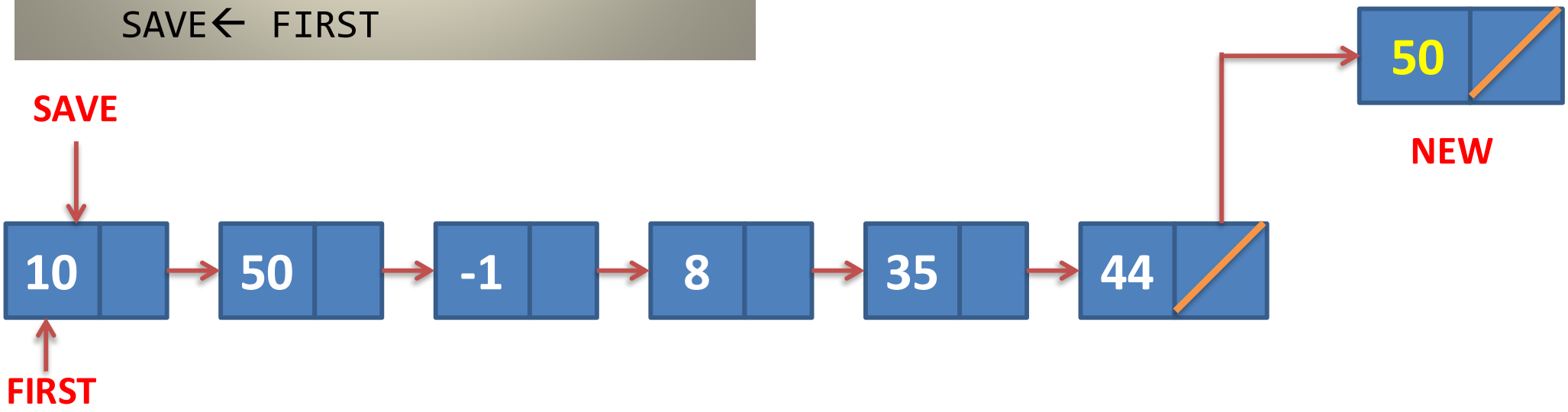
```
1. [Underflow?]
   IF    AVAIL = NULL
   Then  Write ("Availability
         Stack Underflow")
         Return(FIRST)

2. [Obtain address of
    next free Node]
    NEW ← AVAIL

3. [Remove free node from
    availability Stack]
    AVAIL ← LINK(AVAIL)

4. [Initialize fields of
   new node]
    INFO(NEW) ← X
    LINK (NEW) ← NULL
```

```
5. [Is the list empty?]
    If    FIRST = NULL
    Then  Return (NEW)
6. [Initialize search for
   a last node]
    SAVE← FIRST

7. [Search for end of list]
   Repeat while LINK (SAVE) ≠ NULL
       SAVE ← LINK (SAVE)

8. [Set link field of last node
   to NEW]
    LINK (SAVE) ← NEW
9. [Return first node pointer]
    Return (FIRST)
```

Function: INSEND(50, FIRST)

```
4. [Initialize fields of
   new node]
    INFO(NEW) ← X
    LINK (NEW) ← NULL

5. [Is the list empty?]
    If     FIRST = NULL
    Then   Return (NEW)

6. [Initialize search for
   a last node]
    SAVE← FIRST
```

```
7. [Search for end of list]
 Repeat while LINK (SAVE) ≠ NULL
    SAVE ← LINK (SAVE)

8. [Set link field of last node
   to NEW]
    LINK (SAVE) ← NEW

9. [Return first node pointer]
    Return (FIRST)
```

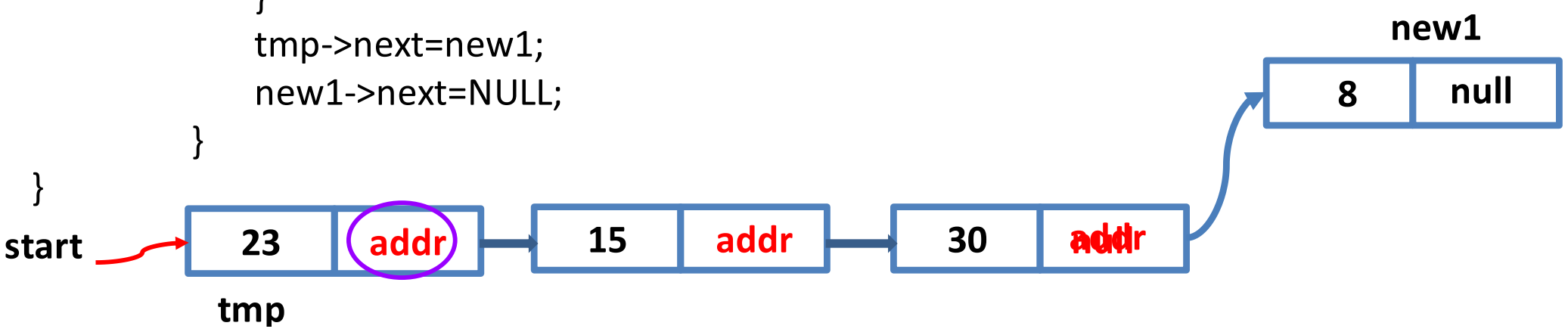


Insert node at end of the given singly link list

```
void insertend()
{
        struct node *new1,*tmp;
        new1=(struct node *)malloc(sizeof(struct node));
        printf(" enter element \n");
        scanf("%d",&new1->d);
        if(start==NULL)
        {
                start=new1;
                new1->next=NULL;
        }
        else
        {       tmp=start;
              while (tmp->next!=NULL)
              {
                      tmp=tmp->next;
              }
              tmp->next=new1;
              new1->next=NULL;
        }
}
```

Condition is false

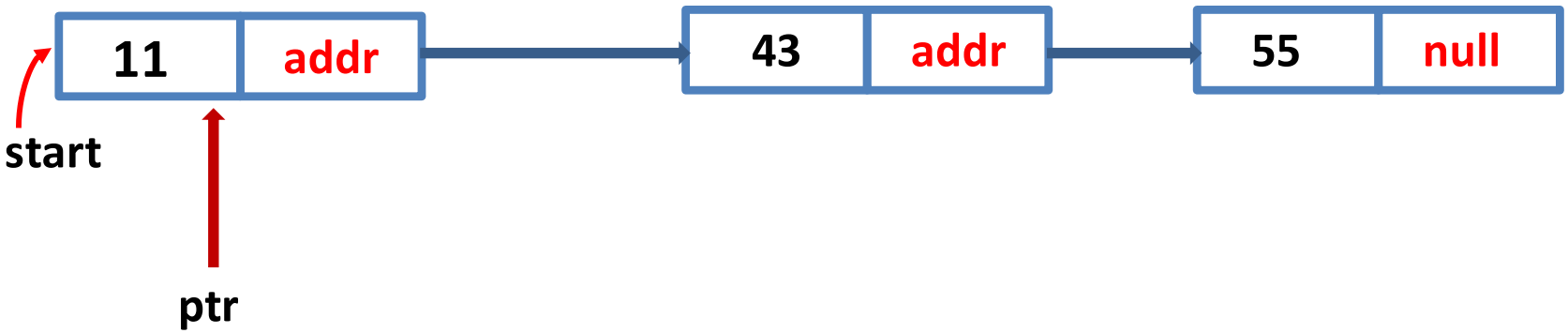
Condition is false



Display given singly link list

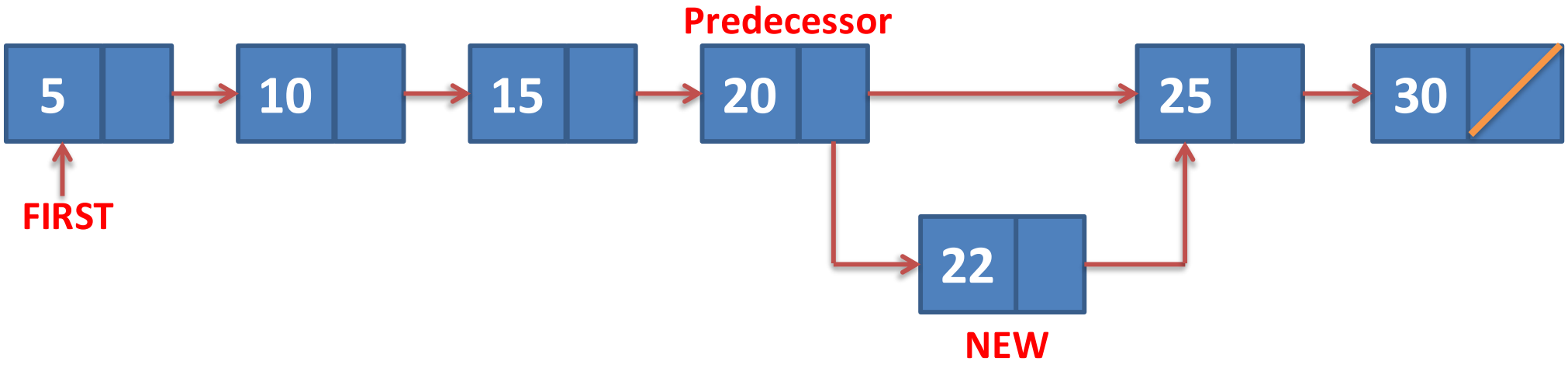
```
void display()
{
        ptr=start;
        while((ptr)!=NULL)   Condition is false
        {
                printf("%d-->",ptr->d);
                ptr=ptr->next;
        }
        printf("null\n");
}
```

Given existing singly link list



Function: INS_AFTER(X, Y, FIRST)

- This function **inserts** a new node after the node having Y in info.
- This function returns address of **FIRST** node.
- **X** is a new element to be inserted.
- **FIRST** is a **pointer to the first element** of a Singly linked linear list.
- Typical node contains **INFO** and **LINK** fields.
- **AVAIL** is a pointer to the top element of the availability stack.
- **NEW** is a temporary pointer variable.



Function: INS_AFTER(X, Y, FIRST)

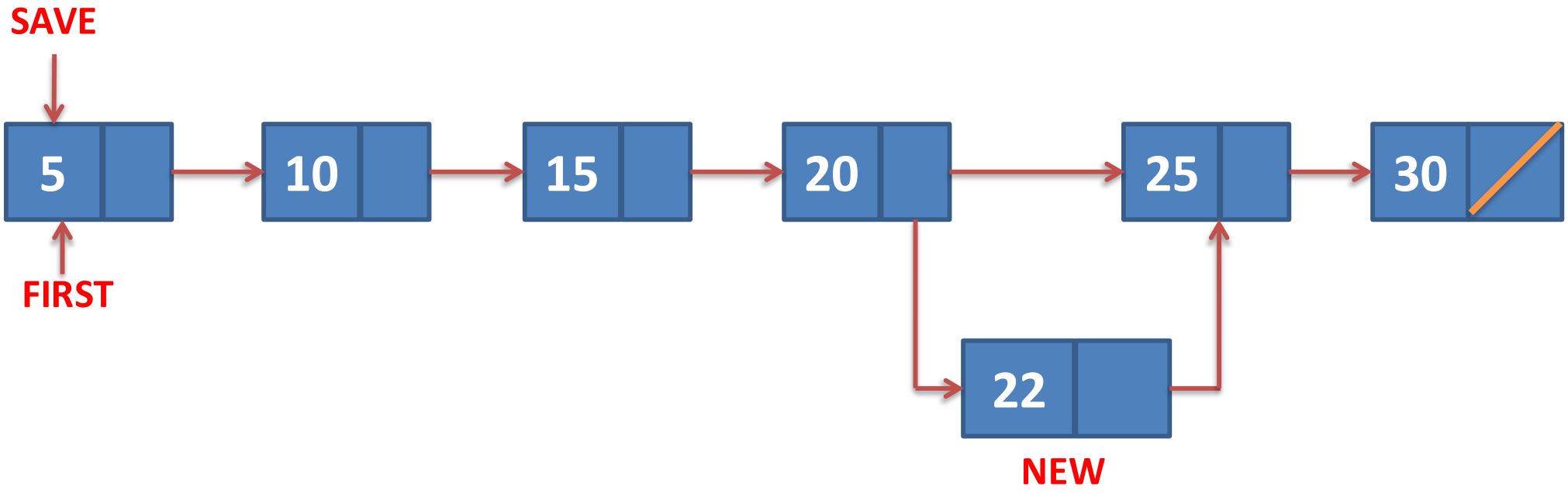
```
1. [Underflow?]
   IF     AVAIL = NULL
   THEN   Write ("Availability
          Stack Underflow")
          Return(FIRST)
2. [Obtain address of
    next free Node]
    NEW ← AVAIL
3. [Remove free node from
    availability Stack]
    AVAIL ← LINK(AVAIL)
4. [Initialize fields of
   new node]
    INFO(NEW) ← X
5. [Is the list empty?]
    IF     FIRST = NULL
    THEN   LINK(NEW) ← NULL
           Return (NEW)
```

```
6. [Initialize temporary pointer]
   SAVE ← FIRST
7. [Search for the node with
info Y]
   Repeat while SINFO(SAVE) != Y
        SAVE ← LINK (SAVE)
8. [Set link field of NEW node
    and its predecessor]
     LINK (NEW) ← LINK (SAVE)
     LINK (SAVE) ← NEW
9. [Return first node pointer]
     Return (FIRST)
```

Function: INS_AFTER(22, 20, FIRST)

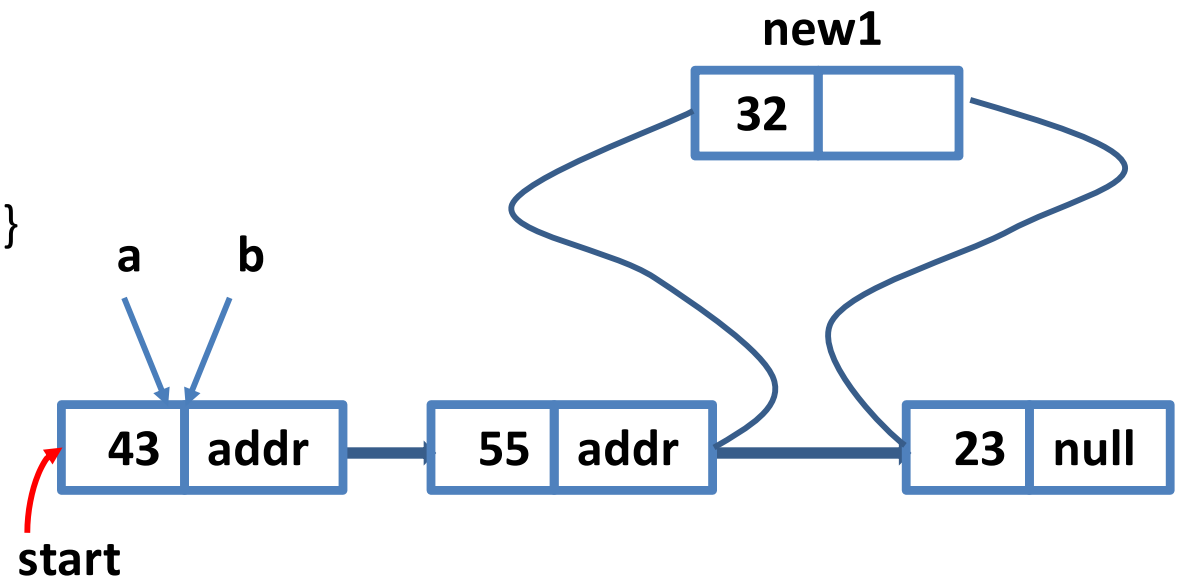
```
6. [Initialize temporary pointer]
   SAVE ← FIRST
7. [Search for the node with
info Y]
   Repeat while INFO(SAVE) != Y
        SAVE ← LINK (SAVE)
```

```
8. [Set link field of NEW node
   and its predecessor]
    LINK (NEW) ← LINK (SAVE)
    LINK (SAVE) ← NEW
9. [Return first node pointer]
    Return (FIRST)
```



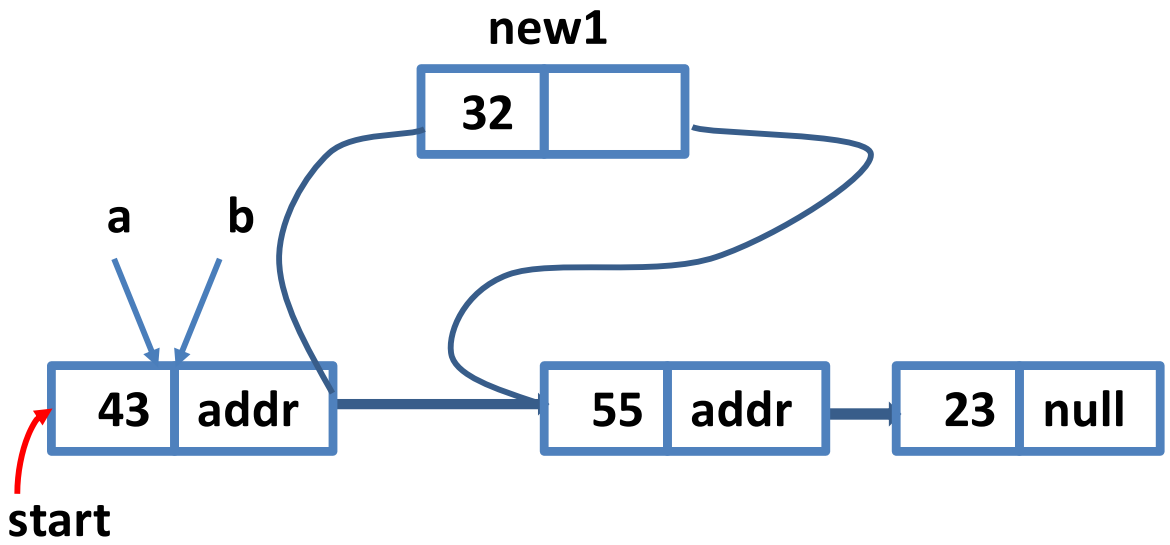
Insert node after given node in the singly link list

```
void insertafter()
{
        int x;
        struct node *new1,*a,*b;
        new1=(struct node *)malloc(sizeof(struct node));
        printf(" enter element \n");
        scanf("%d",&new1->d);
        printf("\nenter a value of a given node after
       you want to insert a new node\n");
        scanf("%d",&x);
        b=a=start;
        if(start==NULL){  False
                printf("empty"); }
        else if(start->d==x){
                a=a->next;
                new1->next=a;
                b->next=new1;}
        else{
                while(b->d!=x){
                        b=a;
                        a=a->next; }//end of while
                b->next=new1;
                new1->next=a; }//end of else}//end of insertafter
```



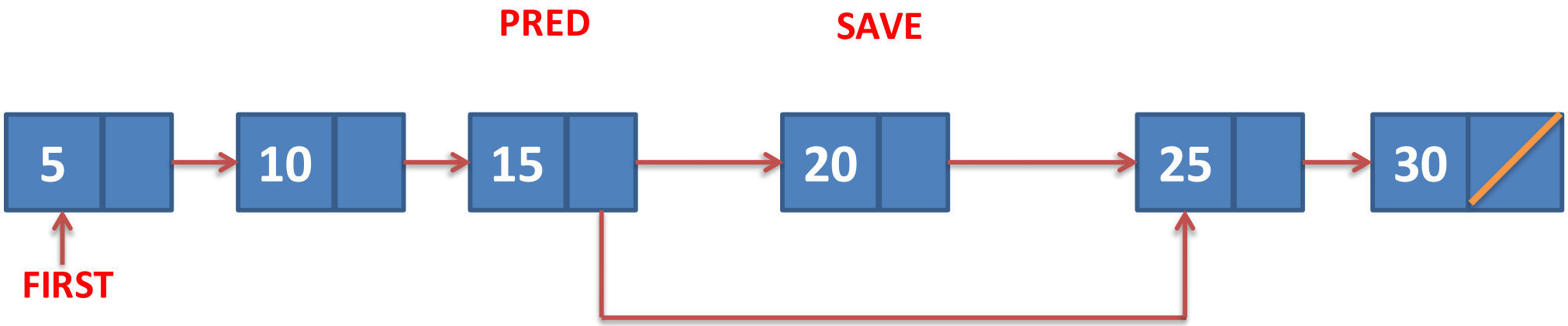
Insert node before given node in the singly link list

```
void insertafter()
{
        int x;
        struct node *new1,*a,*b;
        new1=(struct node *)malloc(sizeof(struct node));
        printf(" enter element \n");
        scanf("%d",&new1->d);
        printf("\nenter a value of a given node before
       you want to insert a new node\n");
        scanf("%d",&x);
        b=a=start;
        if(start==NULL)   False
        {
                printf("empty");
        }


        else{
                while(a->d!=x){
                        b=a;
                        a=a->next; }//end of while
                b->next=new1;
                new1->next=a; }//end of else}//end of insertafter
```



Procedure: DELETE(X, FIRST)

- This algorithm **delete** a node whose address is given by variable **X**.
- **FIRST** is a **pointer to the first element** of a Singly linked linear list.
- Typical node contains **INFO** and **LINK** fields.
- **SAVE & PRED** are temporary pointer variable.



Procedure: DELETE(X, FIRST)

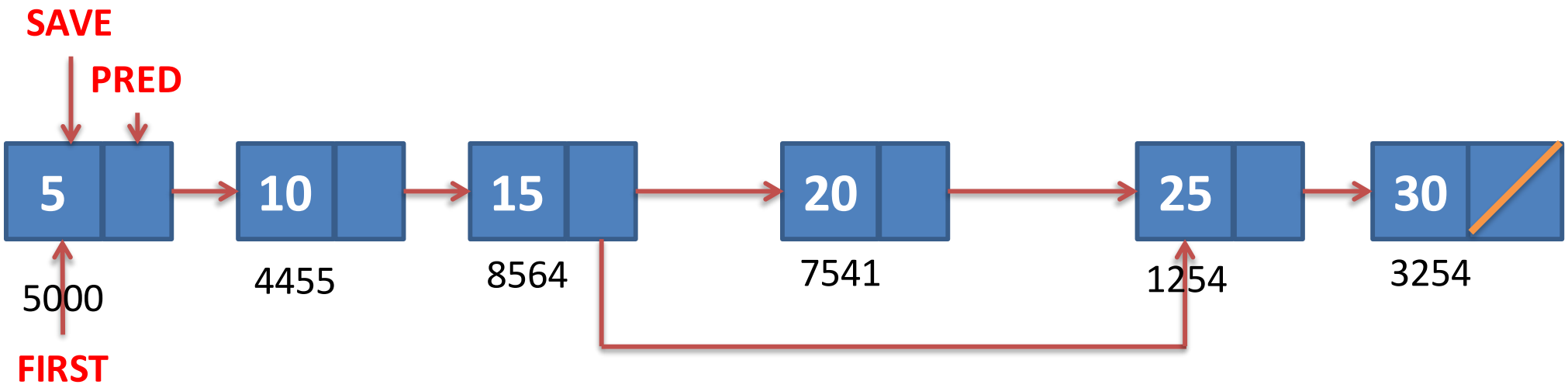
```
1. [Is Empty list?]
IF      FIRST = NULL
THEN    write ('Underflow')
        Return

2. [Initialize search for X]
     SAVE ← FIRST

3. [Find X]
Repeat thru step-5
    while INFO(SAVE) ≠ X and
    LINK (SAVE) ≠ NULL

4. [Update predecessor marker]
     PRED ← SAVE
5. [Move to next node]
     SAVE ← LINK(SAVE)
```

```
6. [End of the list?]
If      INFO(SAVE) ≠ X
THEN    write ('Node not found')
        Return

7. [Delete X]
If   X = INFO(FIRST)
THEN    FIRST ← LINK(FIRST)
ELSE    LINK (PRED) ← LINK (X)

8. [Free Deleted Node]
     Free (X)
```

Procedure: DELETE(20, FIRST)

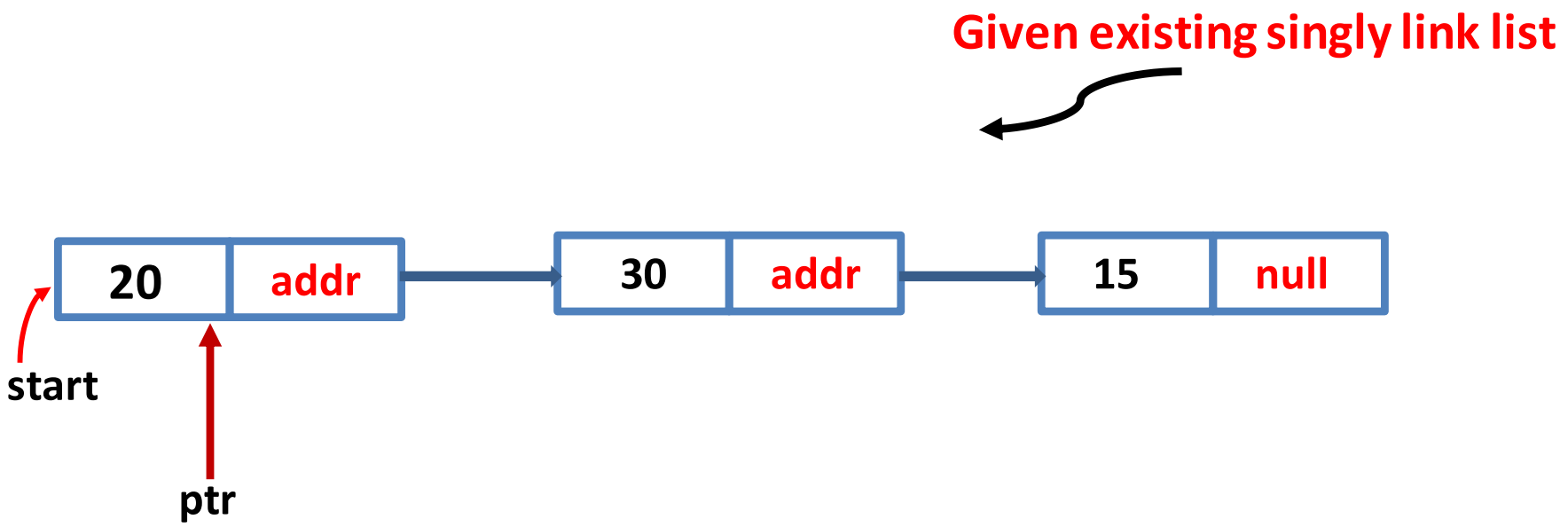
```
2. [Initialize search for X]
     SAVE ← FIRST
3. [Find X]
Repeat thru step-5
    while INFO(SAVE) ≠ X and
    LINK (SAVE) ≠ NULL
4. [Update predecessor marker]
     PRED ← SAVE
5. [Move to next node]
     SAVE ← LINK(SAVE)
```

```
6. [End of the list?]
If      INFO(SAVE) ≠ X
THEN    write ('Node not found')
        Return
7. [Delete X]
If   X = INFO(FIRST)
THEN    FIRST ← LINK(FIRST)
ELSE    LINK (PRED) ← LINK (X)
8. [Free Deleted Node]
     Free (X)
```



Delete first node from the singly link list

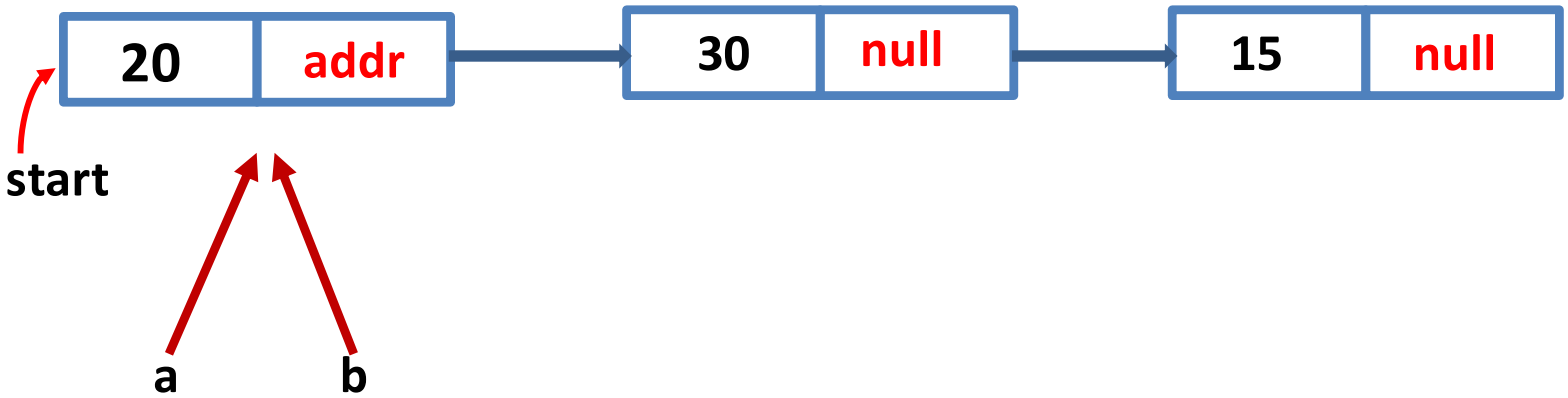
```
void delbegin()
{
    struct node *ptr;
    ptr=start;
    start=ptr->next;
    free(ptr);
}
```



Delete last node from the singly link list

```
void delend()
{
        struct node *a, *b;
        a=b=start;

        while(b->next!=NULL)      False
        {
         a=b;
         b=b->next;
        }
        a->next=NULL;

}
```

Given existing singly link list



Delete selected node from the singly link list

```
void delselected(){
        struct node *a, *b;
        int x;
        a=b=start;
        printf("\nEnter element to be delete\n");
        scanf("%d",&x);

        while(b->d!=x && b->next!=NULL){
         a=b;
         b=b->next;
        }

        if(b->next==NULL){
                if(b->d!=x)
                printf("Element not found");
                else{
                        a->next=NULL;
                }
        }
        else
        a->next=b->next;
}
```